Getting started with the X-CUBE-SUBG1 for Point-to-Point communications based on Sub-1 GHz RF STM32 expansion board

Introduction

X-CUBE-SUBG1 is a demo application for the STM32 to build applications using the Sub-1 GHz RF SPIRIT1 device. It is easily ported across different MCU families thanks to STM32Cube.

This manual describes how to get started with the X-CUBE-SUBG1 software for Point-to-Point (P2P) applications, which represent the most straightforward protocol for bidirectional wireless transmission of data packets from a transmitter to a receiver.

The software provides implementation examples for STM32 Nucleo platforms equipped with the X-NUCLEO-IDS01A4 expansion board based on the SPSGRF-868 module or the X-NUCLEO-IDS01A5 expansion board based on the SPSGRF-915 module.

The following demo examples are available for testing with the evaluation board:

- WM-Bus wireless metering bus demo
- Point-to-Point communication protocol demo

This document explains the Point-to-Point communication demo.

The software is based on STM32Cube technology and expands the STM32Cube-based range of solutions.

Contents

1 STM32Cube 4
1.1 STM32Cube overview 4
1.2 STM32Cube architecture 4

2 X-CUBE-SUBG1 software expansion for STM32Cube 6
2.1 Overview 6
2.2 Architecture 6
2.3 Folders structure 7
2.4 APIs 8

3 Point-to-Point (P2P) demo firmware description 9
3.1 P2P application details 9
3.2 Application state diagram 9
3.3 SPIRIT1 packet handler overview 11
3.4 Transmit and receive packet structure 12
3.4.1 Packet fields and their meaning 13
3.5 User configuration 13
3.5.1 Selecting the SPIRIT1 expansion board platform 13
3.5.2 Selecting packet handler 14
3.5.3 Setting low power mode 14
3.5.4 Setting radio configuration parameters 15
3.5.5 Setting packet configuration parameters 15
3.5.6 Setting address of the nodes 16
3.5.7 User defined commands and macros 16
3.6 Hardware description 16
3.6.1 STM32 Nucleo platform 17
3.6.2 X-NUCLEO-IDS01Ax expansion board 17
3.7 Software description 18
3.8 Hardware setup 18
3.8.1 Hardware setup 18
3.8.2 Setting up the board 19

4 Acronyms and abbreviations 20

5 Reference 21
5.1 Document conventions 21
6 Revision history 22

1 STM32Cube

1.1 STM32Cube overview

The STMCube™ initiative was designed by STMicroelectronics to help developers reduce development effort, time and cost. STM32Cube covers the entire STM32 portfolio. Version 1.x includes:

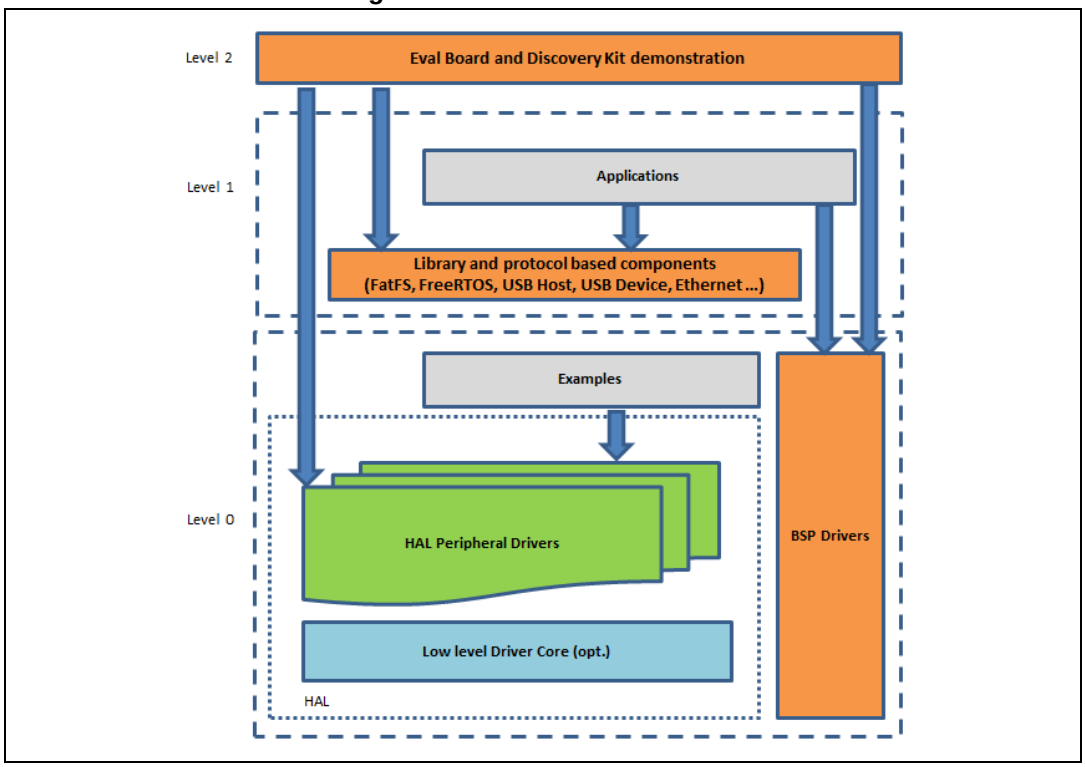
- STM32CubeMX, a GUI for generating C initialization code
- a comprehensive embedded software platform for each series (e.g., STM32CubeF4 for the STM32F4 series):
  - the STM32Cube HAL: STM32 abstraction layer embedded software for maximum portability across the STM32 portfolio
  - a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - all embedded software utilities with a full set of examples

Detailed information regarding STM32Cube is available on st.com at http://www.st.com/stm32cube.

1.2 STM32Cube architecture

The STM32Cube firmware is built around three independent levels that can easily interact with each other, as shown in the figure below.

Figure 1. Firmware architecture



Level 0 is divided into three sub-layers:

- **Board support package (BSP):** offers a set of APIs for the hardware components on the hardware boards (audio codec, IO expander, touchscreen, SRAM driver, LCD drivers, etc.) and consists of two parts:
  - **component**: the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the BSP driver external components and may be ported to any other board
  - **BSP driver**: links the component driver to a specific board and provides a set of user-friendly APIs named BSP_*FUNCT_Action*() (e.g., BSP_LED_Init(), BSP_LED_On(), etc.)

  It is based on modular architecture, which is easily ported to any hardware by simply implementing the low level routines.
- **Hardware Abstraction Layer (HAL):** this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks).
  It provides generic, multi-instance and function-oriented APIs which render user applications unnecessary by providing ready to use processes. For example, for the communication peripherals (I²S, UART, et.), it provides APIs to initialize and configure the peripheral, manage data transfer based on polling, interrupts or DMA processes and manage any communication errors that may raise.
  The HAL Drivers APIs are split into two categories:
  - generic APIs with common and generic functions for the entire STM32 series
  - extension APIs with special functions for a specific family or part number
- **Basic peripheral usage examples:** this layer covers the examples for the STM32 peripheral using the HAL and BSP resources.

Level 1 is divided into two sub-layers:

- **Middleware components:** set of libraries covering USB Host and Device libraries, STemWin, FreeRTOS, FatFS, LwIP and PolarSSL. Horizontal interaction between the components on this layer is performed directly by calling the feature APIs, while the vertical interaction with the low-level drivers is managed through specific callbacks and static macros implemented in the library system call interface.
  For example, the FatFs implements the disk I/O driver to access the microSD drive or the USB Mass Storage Class.
- **Examples based on the middleware components:** each middleware component comes with one or more examples (or ‘applications’) demonstrating how it is used. Integration examples that use several middleware components are also provided.

Level 2 provides comprehensive, real-time and graphical demonstrations of the middleware service layer, the low-level abstraction layer and basic peripheral usage applications for board-based functions.

2 X-CUBE-SUBG1 software expansion for STM32Cube

2.1 Overview

X-CUBE-SUBG1 is a software package that expands the functionality of STM32Cube.

The key features of the package are:

- demo P2P communication example to transfer data from one node to another
- middleware to build applications for WM-Bus (wireless metering bus)
- easy portability across different MCU families thanks to STM32Cube
- free user-friendly license terms
- examples implementations available on board the X-NUCLEO-IDS01A4 (868 MHz) or X-NUCLEO-IDS01A5 (915 MHz) connected onto a NUCLEO-L053R8 or NUCLEO-F401RE

The following further be developments are possible by manipulating the software described in this document:

- automatic meter reading
- home and building automation
- WSN (wireless sensors network)
- industrial monitoring and control
- wireless fire and security alarm systems

The firmware partitioning between the STM32 on Nucleo boards and the SPIRIT1 device is explained below:

STM32 MCU:

- P2P application implementation
- low power mode handling
- interrupt services

SPIRIT1 role:

- basic/STack modes
- header, sync and trailer fields
- encoding/decoding
- sync detection
- TX and RX FIFO

2.2 Architecture

This software is fully compliant with and expands on STM32Cube (see *Section 1: STM32Cube*) to enable development of applications using X-NUCLEO-IDS01Ax (X-NUCLEO-IDS01A4 or X-NUCLEO-IDS01A5) boards hosting the SPIRIT1 device module.

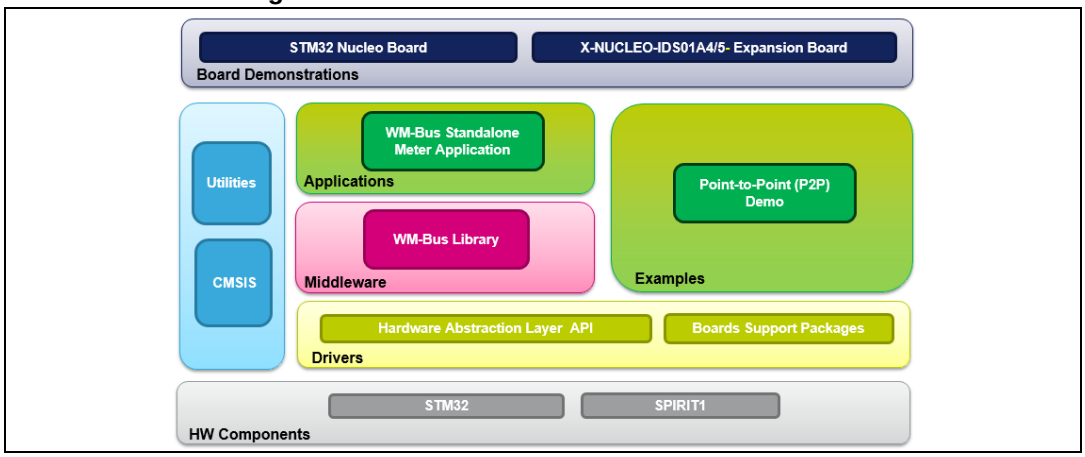
The software is based on STM32CubeHAL, the hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the SPIRIT1 expansion board and some example firmware for P2P communication.

The software layers used by the application software to access and use the SPIRIT1 expansion board are:

- STM32Cube HAL layer: provides a generic, multi-instance set of APIs to interact with the upper layers (the application, libraries and stacks). It consists of generic and extension APIs based on a common architecture which allows other layers like the middleware layer to function without specific Microcontroller Unit (MCU) hardware configurations.
  This structure improves library code reusability and guarantees easy device portability.
- Board support package (BSP) Layer: includes the software to support the peripherals on the STM32 Nucleo board (apart from the MCU). It is a set of APIs which provides a programming interface for certain board-specific peripherals (LED, user button etc.). The BSP firmware layer of the X-NUCLEO-IDS01Ax boards contains APIs for the hardware components and consists of two parts:
  a) **Component**: this is the driver related to the external device on the board and not related to the STM32 (see *Section 1: STM32Cube*). The SPIRIT1 BSP driver is known as the firmware component.
  The SPIRIT1 component driver provides specific APIs and can be ported to and used on any board.
  b) **BSP driver**: enables the component driver to be linked to a specific board and provides a set of user-friendly APIs.
- Middleware: includes the WM-Bus, USB, touch sensing etc. libraries. There is no middleware component For Point-to-Point applications as the demo/application layer interacts with the SPIRIT1 link layer directly
- Application layer: provides a Point-to-Point communication example which involves sending a buffer from one node to another and acknowledgments using the features in the SPIRIT1 link layer.

The figure below outlines the software architecture of the package.

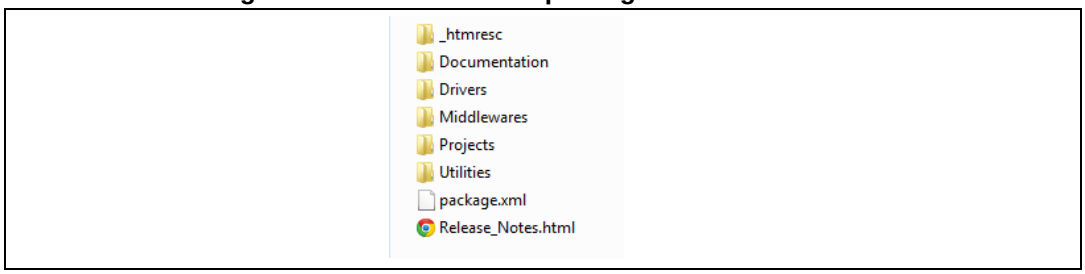
Figure 2. X-CUBE-SUBG1 software architecture



2.3 Folders structure

This section provides an overview of the package folders structure. The next figure outlines the architecture of the package.

Figure 3. X-CUBE-SUBG1 package folders structure



The following folders are included in the software package:

- 'Documentation': contains a compiled HTML file generated from the source code and detailed documentation of the software components and APIs
- 'Drivers': contains the HAL drivers and the board-specific drivers for supported board and hardware platforms, including those for the on-board components and the CMSIS vendor-independent hardware abstraction layer for the Cortex-M processor series
- 'Middlewares': contains libraries for WM-Bus
- 'Projects': contains a sample application used for WM-Bus and P2P firmware examples for the NUCLEO-L053R8 and NUCLEO-F401RE platforms with three development environments (IAR Embedded Workbench for ARM, RealView Microcontroller Development Kit (MDK-ARM), System Workbench for STM32 (SW4STM32)
- 'Utilities': this folder contains a 'PC_software' subfolder with a Windows PC utility for WM-Bus usage and testing. Refer to the relevant documentation.

2.4 APIs

Detailed descritions of all the functions and parameters of the user APIs user can be found in a compiled HTML file located inside the 'Documentation' folder.

3 Point-to-Point (P2P) demo firmware description

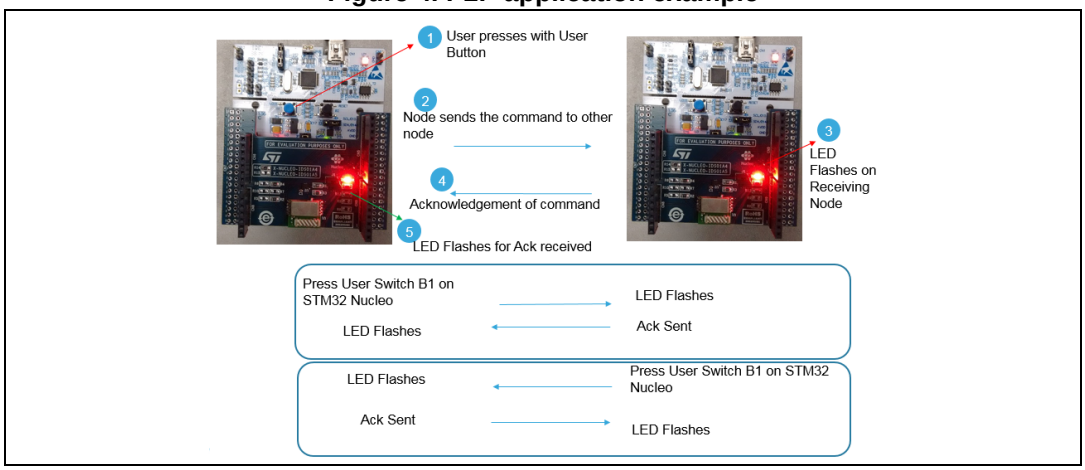
The following section explains how the demo firmware is implemented, the user settings and configurations available and how to modify the firmware for other applications.

3.1 P2P application details

This P2P application is designed to operate thus:

1. on each node (STM32 Nucleo board + SPIRIT1 expansion board), the user button on the Nucleo board can be used to transmit a buffer to the other node.
2. on receiving the signal, the receiver node LED lights up and an acknowledgment (ACK) signal is returned the transmitter node
3. on reception of the ACK signal, the transmitter node LED flashes four times and switches off after a delay period

Figure 4. P2P application example



3.2 Application state diagram

This section explains how to run the demo example with the STM32 Nucleo boards.

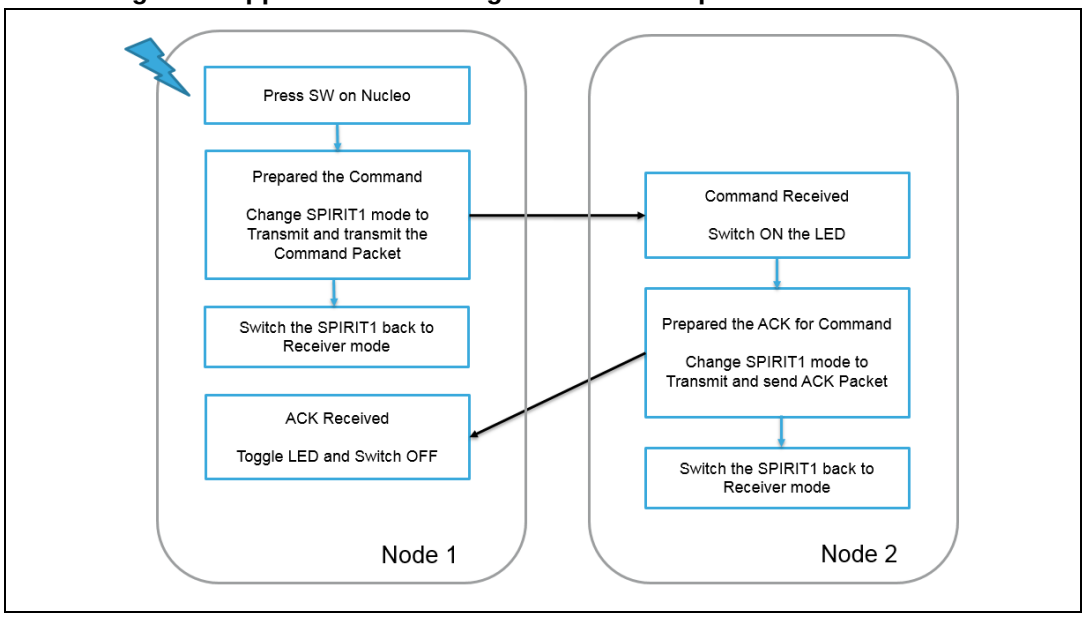
The following figure shows the events and communication flow described above.

The SPIRIT1 remains by default in receive mode but changes to transmit mode when the switch is pressed. Once transmission has terminated, the SPIRIT1 transceiver returns to its default receive mode. On successful completion of the two-way communication (Command/Ack), the MCU enters low-power mode.

To limit low-power mode current consumption, the LED is switched OFF by default. The LED only glows for a short time.

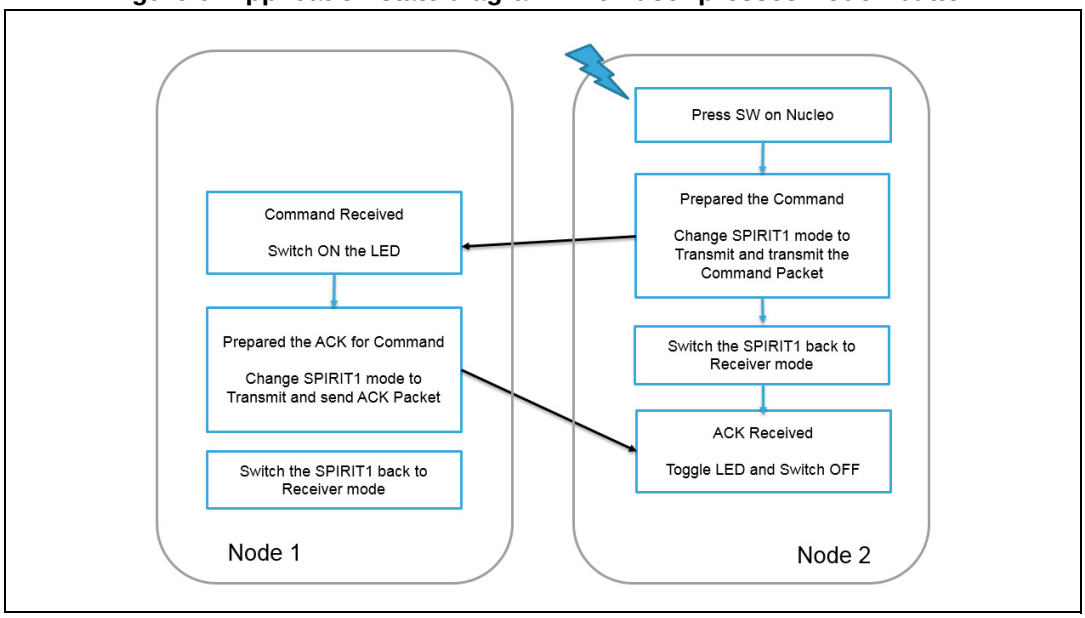
The nodes in P2P have the same functionality and the address of each node is set in the firmware via the user definitions.

Figure 5. Application state diagram when user presses Node1 button


Press SW on Nucleo
Prepared the Command
Change SPIRIT1 mode to Transmit and transmit the Command Packet
Switch the SPIRIT1 back to Receiver mode
ACK Received
Toggle LED and Switch OFF
Node 1
Command Received
Switch ON the LED
Prepared the ACK for Command
Change SPIRIT1 mode to Transmit and send ACK Packet
Switch the SPIRIT1 back to Receiver mode
Node 2

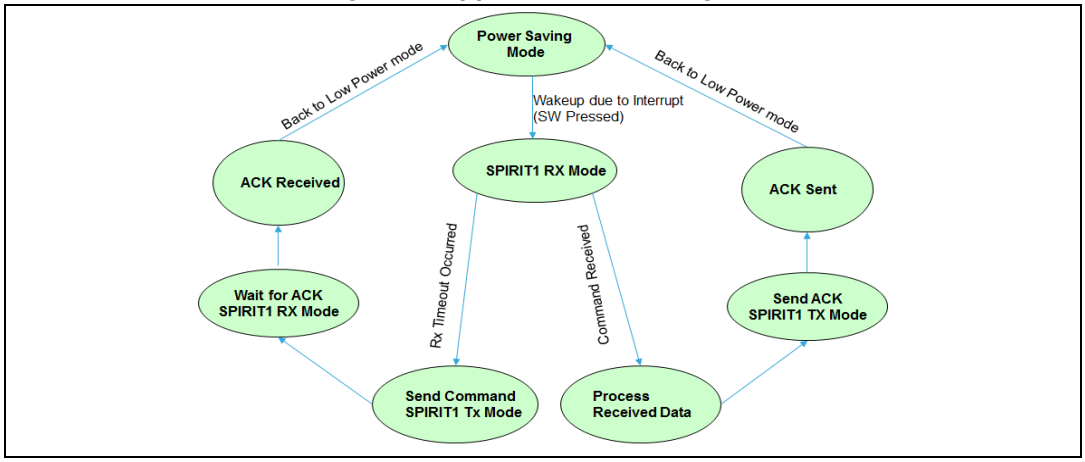
If the user decides to press the switch on the other node, the functionality remains the same. The Node will wake-up from low power mode, prepare the command for transmission and send the data packet. Then it will wait for acknowledgment.

Figure 6. Application state diagram when user presses Node2 button


Command Received
Switch ON the LED
Prepared the ACK for Command
Change SPIRIT1 mode to Transmit and send ACK Packet
Switch the SPIRIT1 back to Receiver mode
Node 1
Press SW on Nucleo
Prepared the Command
Change SPIRIT1 mode to Transmit and transmit the Command Packet
Switch the SPIRIT1 back to Receiver mode
ACK Received
Toggle LED and Switch OFF
Node 2

The following diagram shows both the transmit and receive states for data communication in the firmware under low power modes.

Figure 7. Application state diagram



3.3 SPIRIT1 packet handler overview

This section explains the SPIRIT1 packet handler. Before on-air transmission, raw data is arranged in a packet structure. The SPIRIT1 offers a highly flexible and fully programmable packet: the structure of the packet, the number, the type, and the dimension of the fields inside the packet depend on one of the possible configuration settings.
Through a suitable register, the user can choose from one of the formats shown in the following tables:

Table 1. STack

Preamble	Sync	Length	Destination address	Source address	Control	Seq. no.	No ACK	Payload	CRC

Table 2. WM-Bus

Preamble	Sync	Payload	Postamble

Table 3. Basic

Preamble	Sync	Lenght	Destination address	Control	Payload	CRC

A detailed description of the SPIRIT1 embedded packet handler is given in the SPIRIT1 datasheet.

Since the P2P communication requires the destination address of the receiving node, the P2P demo is based on stack and basic packet handlers. The WM-Bus packet format is not used and is beyond the scope of this example demonstration.

A basic comparison of the different packet handlers is given below.

Table 4. Packet handler feature comparison

Features	STack	WM-Bus	Basic
Destination address filtering	Yes	No	Yes
Broadcast and multicast addressing	Yes	No	Yes
Source address filtering	Yes	No	No
Custom filtering	Yes	No	Yes
CRC filtering	Yes	No	Yes
(1)LLP: automatic acknowledgment	Yes	No	No
(1)LLP: automatic acknowledgment with piggybacking	Yes	No	No
(1)LLP: automatic retransmission	Yes	No	No

1. Link layer protocol

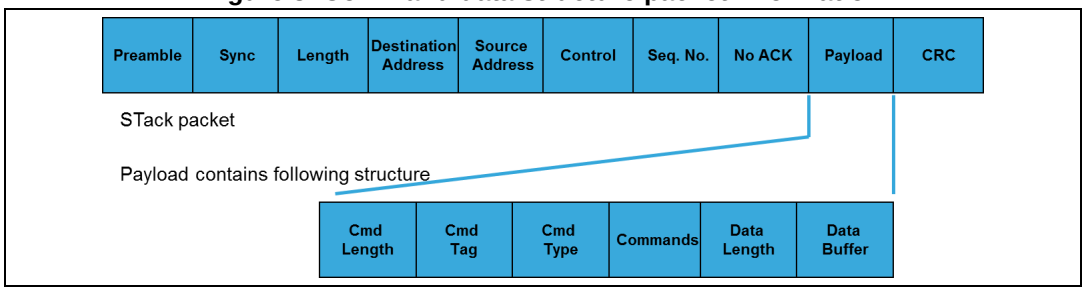
3.4 Transmit and receive packet structure

This section explains the command and response packet structure.

Command packet features:

- command with data
- SPIRIT1 is able to handle 65535 bytes of data
- avoid sending the command and data separately
- user defines own command structure
- user defines maximum data packet size

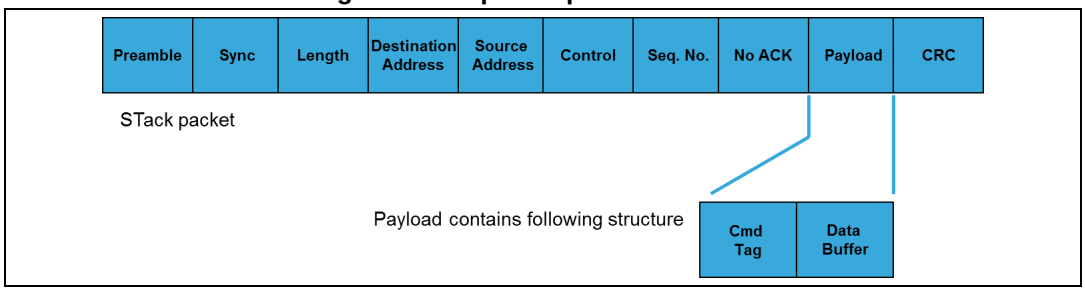
Figure 8. Command data structure packet information



Response packet features:

- data buffer is replied from the node
- tag contains the number associated with the command so the receiver can associate the response with the specific command

Figure 9. Response packet structure



3.4.1 Packet fields and their meaning

Cmd Length: the basic command can be 1 byte long, however the user can have multiple command bytes

Cmd Tag: a unique tag number is attached to each command issued from node and the response must replicate this same number

Cmd Type: flag for identifying application level or network command

Commands: the actual command set sent from the source to destination, may include parameters

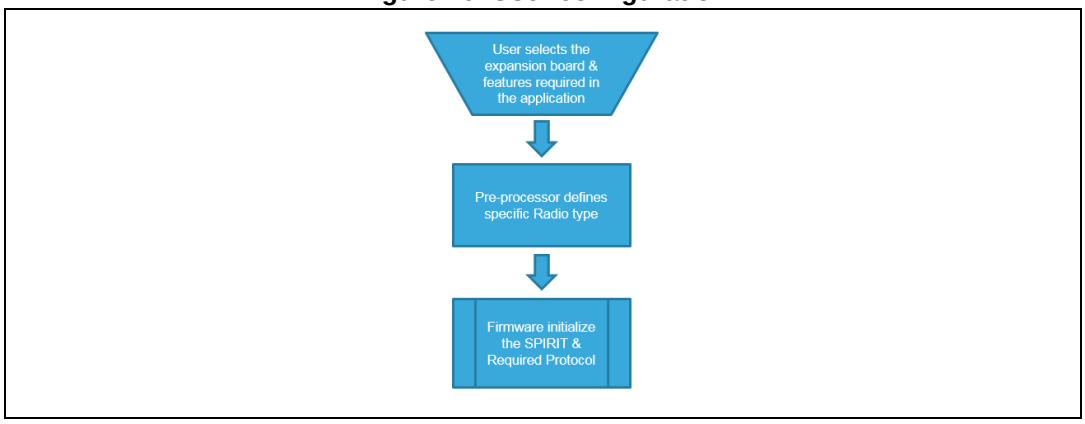
Data Length: the size of the data packet length

Data buffer: the actual data associated with the command

3.5 User configuration

This section explains the configuration file (spirit1_appli.h) that can be modified by the user according to the application.

Figure 10. User configuration


User selects the expansion board & features required in the application
Pre-processor defines specific Radio type
Firmware initialize the SPIRIT & Required Protocol

The different sections of the user configurable file are described below with the default settings.

3.5.1 Selecting the SPIRIT1 expansion board platform

The user can select the desired SPIRIT1 expansion board platform by uncommenting the macros shown below. Once the SPIRIT1 expansion board platform is selected, the operating frequency is handled by firmware itself.

For example, if X-NUCLEO-IDS01A4 is selected as the SPIRIT1 expansion board, the operating frequency selected by the firmware is 868 MHz.

Figure 11. Selecting SPIRIT1 expansion board platform in the firmware

```
/*  Platform definition : Uncomment the used expansion board */

#define X_NUCLEO_IDS01A4
// #define X_NUCLEO_IDS01A5

#if defined(X_NUCLEO_IDS01A4)
#define USE_SPIRIT1_868MHz
#elif defined(X_NUCLEO_IDS01A5)
#define USE_SPIRIT1_915MHz
#else
#error SPIRIT1 Nucleo expansion board undefined or unsupported
#endif
```

3.5.2 Selecting packet handler

The user can select the desired features by uncommenting the relevant macros below. By default, the SPIRIT1 works with BASIC packet handler.

SPIRIT1 will only use the STack packet handler if link layer features like auto ack, piggybacking and auto retransmission are defined.

Figure 12. Selecting packet handler in the firmware

```
/*  Uncomment the Link Layer features to be used */
// #define USE_AUTO_ACK
// #define USE_AUTO_ACK_PIGGYBACKING
// #define USE_AUTO_RETRANSMISSION

#if defined(USE_AUTO_ACK)&& defined(USE_AUTO_ACK_PIGGYBACKING)&&
defined(USE_AUTO_RETRANSMISSION)
#define USE_STack_PROTOCOL

/* LLP configuration parameters */
#define EN_AUTOACK                  S_ENABLE
#define EN_PIGGYBACKING             S_ENABLE
#define MAX_RETRANSMISSIONS             PKT_N_RETX_2

#else
#define USE_BASIC_PROTOCOL
#endif
```

3.5.3 Setting low power mode

The P2P application supports low power mode, which is enabled by default. It is implemented in such a way as to allow the MCU to either enter STOP mode or SLEEP mode.

SPIRIT1 can be set to STANDBY mode to lower power consumption further.

Figure 13. Setting low power mode in the firmware

```
/* Uncomment the system Low Power Operating mode */
#define USE_LOW_POWER_MODE

#if defined (USE_LOW_POWER_MODE)
#define LPM_ENABLE
//#define MCU_STOP_MODE
#define MCU_SLEEP_MODE
//#define RF_STANDBY
#endif
```

3.5.4 Setting radio configuration parameters

The user can set the radio parameter in the following section of the configuration file, even though changing these settings is not recommended.

Figure 14. Setting setting radio configuration parameters in the firmware

```
/* Radio configuration parameters */
#define XTAL_OFFSET_PPM          0
#define INFINITE_TIMEOUT                0.0

#ifdef USE_SPIRIT1_433MHz
#define BASE_FREQUENCY           433.0e6
#endif
#ifdef USE_SPIRIT1_868MHz
#define BASE_FREQUENCY           868.0e6
#endif
#ifdef USE_SPIRIT1_915MHz
#define BASE_FREQUENCY           915.0e6
#endif

#define CHANNEL_SPACE            20e3
#define CHANNEL_NUMBER           0
#define MODULATION_SELECT        FSK
#define DATARATE                        38400
#define FREQ_DEVIATION                  20e3
#define BANDWIDTH                       100E3
#define POWER_DBM                       11.6
#define POWER_INDEX                     7
#define RECEIVE_TIMEOUT          2000.0 /*change the value for required timeout period*/
```

3.5.5 Setting packet configuration parameters

The user can set the packet configuration in following part of code, even though changing these settings is not recommended.

Figure 15. Setting packet configuration parameters in the firmware

```
/* Packet configuration parameters */
#define PREAMBLE_LENGTH                 PKT_PREAMBLE_LENGTH_04BYTES
#define SYNC_LENGTH                     PKT_SYNC_LENGTH_4BYTES
#define SYNC_WORD                       0x1A2635A8
#define LENGTH_TYPE                     PKT_LENGTH_VAR
#define LENGTH_WIDTH                    7
#define CRC_MODE                        PKT_CRC_MODE_8BITS
#define CONTROL_LENGTH                  PKT_CONTROL_LENGTH_0BYTES
#define EN_ADDRESS                      S_DISABLE
#define EN_FEC                          S_DISABLE
```

3.5.6 Setting address of the nodes

Addresses can be set in following section of the system setup guide.

Figure 16. Address setting for nodes

```
/* Addresses configuration parameters */
#define EN_FILT_MY_ADDRESS              S_DISABLE
#define MY_ADDRESS                      0x34
#define EN_FILT_MULTICAST_ADDRESS       S_DISABLE
#define MULTICAST_ADDRESS               0xEE
#define EN_FILT_BROADCAST_ADDRESS       S_DISABLE
#define BROADCAST_ADDRESS               0xFF
#define DESTINATION_ADDRESS             0x44
#define EN_FILT_SOURCE_ADDRESS          S_DISABLE
#define SOURCE_ADDR_MASK                0xf0
```

3.5.7 User defined commands and macros

Below are the user defined commands and macros.

Figure 17. Setting user defined commands and macros in the firmware

```
/* User Command */
#define APPLI_CMD                       0x11
#define NWK_CMD                         0x22
#define LED_TOGGLE                      0xff
#define ACK_OK                          0x01
#define MAX_BUFFER_LEN                  96
#define TIME_TO_EXIT_RX                 3000
#define DELAY_RX_LED_TOGGLE             200
#define DELAY_TX_LED_GLOW               1000
#define LPM_WAKEUP_TIME                 100
```

3.6 Hardware description

This section describes the hardware components required for SPIRIT1-based applications.

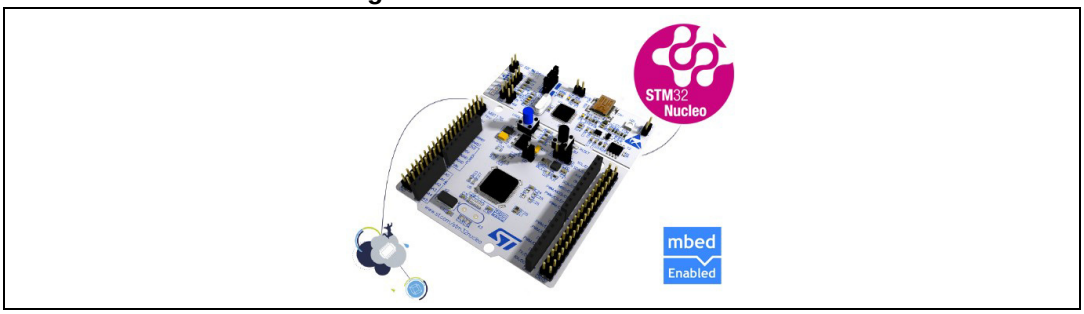
For further details, refer to document *UM1872: Getting started with the Sub-1 GHz expansion board based on the SPSGRF-868 and SPSGRF-915 modules for STM32*.

3.6.1 STM32 Nucleo platform

The STM32 Nucleo boards provide an affordable and flexible way for users to try out new ideas and build prototypes with any STM32 microcontroller lines. The Arduino™ connectivity support and ST Morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized expansion boards. The STM32 Nucleo board does not require any separate probes as it integrates the ST-LINK/V2-1 debugger/programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples.

Information regarding STM32 Nucleo boards is available on st.com at http://www.st.com/stm32nucleo.

Figure 18. STM32 Nucleo board

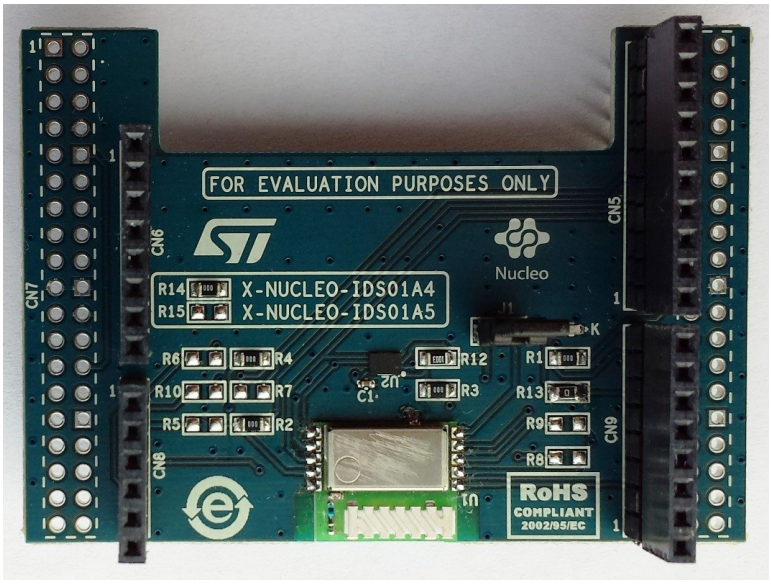


3.6.2 X-NUCLEO-IDS01Ax expansion board

The X-NUCLEO-IDS01Ax is a demonstration kit for evaluating the features and capabilities of SPSGRF module based on the SPIRIT1 low data rate, low power sub-1 GHz transceiver device.

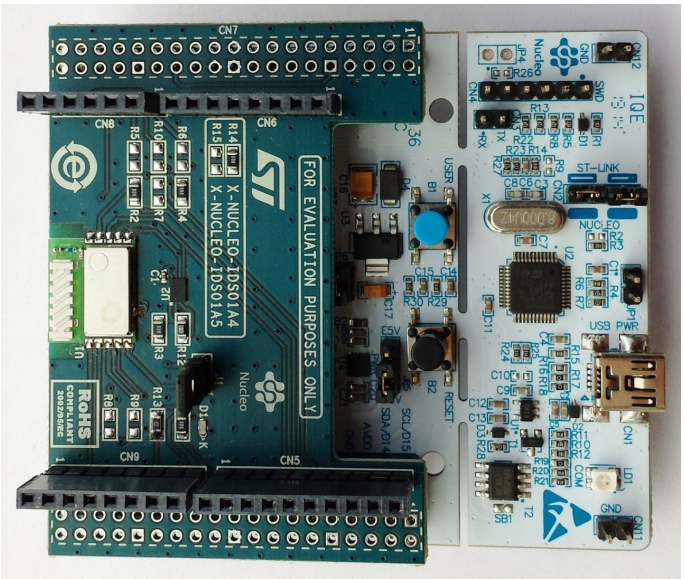
The expansion board includes on board SPI EEPROM for saving parameters and user interface LED.

Figure 19. X-NUCLEO-IDS1Ax expansion board



Information regarding the X-NUCLEO-IDS01A4 and X-NUCLEO-IDS01A5 STM32 expansion boards is available on st.com at http://www.st.com/x-nucleo.

Figure 20. X-NUCLEO-IDS1Ax expansion board plugged to STM32 Nucleo board



3.7 Software description

The following software components establish the development environment for creating applications using the STM32 Nucleo equipped with a SPIRIT1 expansion board:

- X-CUBE-SUBG1: an STM32Cube expansion for SPIRIT1 application development. The X-CUBE-SUBG1 firmware and relative documentation is available on st.com.
- development tool-chain and compiler: the STM32Cube expansion software supports the following environments:
  - IAR Embedded Workbench for ARM® (EWARM) toolchain + ST-Link
  - RealView Microcontroller Development Kit (MDK-ARM) toolchain + ST-LINK
  - System Workbench for STM32 (SW4STM32) + ST-LINK

3.8 Hardware setup

This section describes the hardware and software setup procedures and the corresponding system setup.

3.8.1 Hardware setup

The following hardware components are needed per node:

1. a STM32 Nucleo Development platform (suggested order code: either NUCLEO-F401RE or NUCLEO-L053R8)
2. a SPIRIT1 expansion board (order code: X-NUCLEO-IDS01A4 (868 MHz) or X-NUCLEO-IDS01A5 (915 MHz))
3. a USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC

Two nodes are needed to replicate the P2P demo.

3.8.2 Setting up the board

Follow these steps to set up the board:

1. check that a jumper is present on the J1 connector. This jumper provides the required voltage to the devices on the board
2. connect the X-NUCLEO-IDS01Ax to the Nucleo board from the top as shown in *Figure 20*
3. power the Nucleo board using the Mini-B USB cable
4. program the firmware in the STM32 on the Nucleo board using the firmware example provided
5. press the reset button on the Nucleo board
6. the demonstration kit is ready to be used

4 Acronyms and abbreviations

Table 5. Acronyms and abbreviations

Acronym	Description
AMR	Automatic meter reading
BSP	Board support package. Generally referred to the hardware interface layer.
EEPROM	Electrically erasable programmable read only memory
GHz	Giga Hertz
GUI	Graphical user interface
HAL	Hardware abstraction layer
LED	Light emitting diode
MCU	Microcontroller unit
P2P	Point-to-Point communication
RF	Radio frequency communication
SPI	Serial peripheral interface
USB	Universal serial bus
WM-Bus	Wireless metering bus
WSN	Wireless sensors network

5 Reference

1. SPIRIT1 device datasheet
2. SPSGRF module datasheet
3. STM32 and Nucleo board datasheets
4. UM1872: Getting started with the Sub-1 GHz expansion board based on the SPSGRF-868 and SPSGRF-915 modules for STM32

5.1 Document conventions

1. X-NUCLEO-IDS01Ax: refers to the SPIRIT1 module SPSGRF expansion board (X-NUCLEO-IDS01A4/ X-NUCLEO-IDS01A5)
2. SPIRIT1 module expansion boards: refers to X-NUCLEO-IDS01Ax (x = 4, 5) as explained above
3. STM32 Nucleo board: refers to NUCLEO-L053R8 or NUCLEO-F401RE unless otherwise specified
4. the firmware described in this document is developed and tested using the NUCLEO-L053R8 or NUCLEO-F401RE and X-NUCLEO-IDS01Ax (x = 4, 5) boards; the demo firmware is easily ported to other NUCLEO or SPIRIT1 boards by making changes to the BSP layer
5. Node: refers to the combination of the STM32 Nucleo board and X-NUCLEO-IDS01Ax expansion board in a connected network

6 Revision history

Table 6. Document revision history

Date	Revision	Changes
10-Jun-2015	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved